VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Stanislav Baričák

Bakalářská práce

Vedoucí práce: Ing. Jan Kožusznik, Ph.D.

Ostrava, 2021

Abstrakt

Cieľom tejto práce bolo oboznámiť o priebehu realizácie odbornej praxe, ktorá prebiehala vo firme Marelli Automotive Lighting Jihlava (Czech Republic) s.r.o. na R&D oddelení. Hlavným cieľom praxe bolo vytvoriť webovú aplikáciu, ktorá bude využívať na evidenciu softvérových licencií. Logická časť bola implementovaná hlavne v jazyku Java EE, pomocou Spring frameworku. Pre úložisko dát v databáze sa použila technológia MySQL. Užívateľské prostredie bolo vytvorené pomocou HTML, CSS a JavaScritpt. Bakalárska práca zahŕňa popis použitých technológii a jazykov, ktoré boli využívané pri vývoji aplikácie a v neposlednom rade zhodnotenie získaných znalostí, nadobudnutých počas priebehu trvania praxe.

Kľúčové slová

Marelli Automotive Lighting Jihlava (Czech Republic) s.r.o., webová aplikácia, databáza, individuálna odborná prax, MySQL, Spring, Thymeleaf

Abstract

The main purpose of this work was to inform about the process of realization of individual professional practice, which took place in the Marelli Automotive Lighting Jihlava (Czech Republic) Ltd company in a R&D department. The main goal of the individual professional practice was to create a web application, which will be used to manage software licences. Logical part was implemented mainly in the Java EE language with help of the Spring framework. For a data storage in database the MySQL technology was used. User interface was created by using HTML, CSS and JavaScript. The bachelor thesis include a description of used technologies and languages, which were used for the application development and last but not least evaluation of gained knowledges acquired during the process of individual professional practice.

Keywords

Marelli Automotive Lighting Jihlava (Czech Republic) Ltd, web application, database, individual professional practice, MySQL, Spring, Thymeleaf

Poďakovanie

Chcel by som poďakovať spoločnosti Marelli Automotive Lighting Jihlava (Czech Republic) s.r.o. za dovolenie vykonávať odbornú prax a hlavne oddeleniu R&D, v ktorom som túto prax vykonával.

Poďakovať chcem aj Ing. Jan Kožusznik, Ph.D., ktorý bol môj vedúci a korigoval môj postup pri tvorbe tejto bakalárskej práce.

Obsah

Zoznam použitých skratiek a symbolov

API	–	Application Programming Interface
CSS	–	Cascading Style Sheet
CSV	–	Comma−separated values
HTML	–	Hypertext Markup Language
HTTP	–	Hypertext Transfer Protocol
MVC	–	Model−View−Controller
POM	–	Project Object Model
R&D	–	Research and development
REST	–	Representational State Transfer
SCSS	–	Sassy Cascading Style Sheet
URI	–	Universal Resource Identifier
URL	–	Uniform Resource Locator

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov zdrojového kódu

Kapitola 1

Úvod

Cieľom tejto bakalárskej práce je popísať priebeh odbornej praxe, počas ktorej som mal navrhnúť a implementovať informačný systém vo firme Marelli Automotive Lighting Jihlava (Czech Republic) s.r.o. na R&D oddelení.

O možnosti vykonať odbornú prax v tejto spoločnosti som sa dozvedel prostredníctvom mojich predošlých pracovných skúseností v tejto firme, ktoré sa bohužiaľ netýkali oboru môjho štúdia. Túto odbornú prax som si zvolil hlavne z dôvodu možnosti získať praktické skúsenosti v oblasti informačných technológií a teda rozšíriť moje znalosti nadobudnuté počas štúdia na VŠB.

V úvodnej časti tejto práce popisujem firmu, v ktorej som túto prax vykonával spolu s mojím pracovným zaradením. Následne popisujem problém, prečo ponuka tejto pracovnej pozície v rámci bakalárskej praxe vznikla. Po predstavení problému uvádzam cieľ praxe, spolu s popisom zadaných úloh počas jej vykonávania. Keďže v tomto projekte bolo využitých mnoho technológii, tak časť tejto práce sa venuje ich popisu, aby čitateľ získal stručný prehľad.

Potom čo je čitateľ oboznámený so spoločnosťou, zadaním a technológiami sa prvá časť sa venuje popisu návrhu a vytvorenia potrebného úložiska dát, bez ktorého by tento projekt nemohol korektne fungovať.

Ďalej popisujem typ architektúry, na ktorej je tento projekt založený spolu s potrebnými závislosťami, ktoré som použil. Následne modely, ktoré mapujú navrhnuté a vytvorené tabuľky v databáze. Potom dôležité prvky každej aplikácie a to validáciu a zabezpečenie aplikácie. Nevynechávam ani popis veľmi dôležitých prvkov tejto webovej aplikácie a to kontrolérov a pohľadov. V neposlednom rade sa venujem popisu môjho posledného zadania, ktoré slúžilo na vytvorenie použiteľných výstupov, ktoré budú využívané oddelením R&D.

V poslednej časti zhodnocujem moje chýbajúce a získané znalosti, spolu s časovou náročnosťou tejto bakalárskej praxe.

Kapitola 2

Predstavenie firmy a pracovnej pozície

2.1 Popis spoločnosti

Pôvodne bola spoločnosť Marelli Automotive Lighting Jihlava v Českej Republike založená v roku 1997 pod firmou Bosch. Vo firme došlo k zmene, keď v roku 1999 vstúpila do nej spoločnosť Magneti Marelli a stala sa jej výhradným vlastníkom od roku 2003.

V spoločnosti Magneti Marelli sa prvého mája 2019 jej divízia, čiže firma Automotive Lighting, zlúčila so spoločnosťou Calsonic Kansei. Keďže sa môže pýšiť dlhou históriou pôsobenia už v 15 krajinách v oblasti výroby inovatívnych technologických súčastiach, ktoré sú hlavne určené pre automobilový priemysel. Keďže sa zmenil vlastník firmy, tak to viedlo k zmene názvu na Marelli Automotive Lighting Jihlava.

Činnosť Marelli Automotive Lighting Jihlava je sústredený na vývoj a výrobu najmodernejších predných svetlometov. Návrh svetlometov prebieha spoločne so zákazníkmi a partnermi vďaka, ktorým sa dotvoria charakteristické prvky určitej značky a modelu. Vďaka kvalitnej sériovej výrobe sú výsledné svetlomety dodávané popredným svetovým výrobcom automobilov a získavajú ocenenia na svetovom trhu.

Firma sa riadi a dodržuje hodnotami, ktoré hlása:

- byť hrdý na značku Marelli Automotive Lighting Jihlava a jej výrobky
- odvaha prekonávať prekážky a ochota hľadať riešenie
- priama a otvorená komunikácia
- spolupracovať tímovo

2.2 Popis pracovnej pozície

Pracovná pozícia vo firme, ktorú som zastával by sa dala najlepšie vyjadriť ako programátor webových aplikácií.

Zadávané úlohy sa postupne stávali zložitejšie a obťažnejšie v priebehu praxe. Z tohto dôvodu som mohol prípadné chýbajúce vedomosti doplniť samoštúdiom . Prípadné otázky som mohol dávať kolegom, hlavne aby som si ujasnil prípadné nejasnosti v požiadavkách na aplikáciu, avšak hlavné slovo o finálnom vzhľade a funkciách aplikácie mal môj konzultant vo spoločnosti.

Kapitola 3

Zadané úlohy a ich popis v priebehu odbornej praxe

Za prvé je potrebné poznať aký problém má aplikácia riešiť, aby bolo možné vytvoriť adekvátne riešenie. V tejto kapitole vám predstavím a popíšem zadané úlohy.

3.1 Problém

Každá firma potrebuje nejakým spôsobom evidovať svoje licenčné výdaje a prípadne zabezpečiť ich dôslednú archiváciu. Zavedenie systému je potrebné z dôvodu veľkého množstva spravovaných licencií, aby sa sprehľadnili licenčné výdaje. Taktiež je potrebné efektívnejšie spracovávať dáta pre prípadné generovanie reportov.

3.2 Hlavná úloha praxe

Mojím cieľom v tejto firme bolo vytvoriť informačný systém pre správu licencií. V tejto práci sa budem venovať riešeniu evidencie a archivácie licencií pre oddelenie R&D. Tento cieľ pozostával z viacerých menších úloh. S technológiami, v ktorých mala byť vytvorená aplikácia som predtým nemal žiadne skúsenosti a stúpala im náročnosť, takže som musel využívať samoštúdium.

3.3 Prvé zadanie

Mojím prvým zadaním bolo zabezpečiť potrebné úložisko, kde bude možné ukladať informácie. Pre požadované fungovanie aplikácie je potrebné korektne navrhnúť štruktúru tabuliek v databáze, aby sa umožnila správna funkčnosť aplikácie. Samotné licencie by mali obsahovať odkaz od dodávateľa, aby bolo možné sa ľahko dostať k softvéru. Vo firme budú zodpovedné osoby k daným licenciám, preto bude potrebné zabezpečiť ich prepojenie s dodávateľmi. Každá licencia sa vždy kupuje na

vymedzený čas, preto bude potrebné zariadiť evidovanie exspirácií daných licencií, avšak keďže sa ceny softvérov môžu zmeniť, bolo by vhodné sa zamerať aj na tento faktor. Taktiež by sa malo zabezpečiť, aby všetci užívatelia mali prístup ku všetkému, preto bude potrebné ich rozdeliť do prístupových skupín a daným skupinám povoliť alebo zakázať funkcie.

3.4 Druhé zadanie

Ako druhé zadanie som dostal vytvoriť hlavnú časť aplikácie. Tá by mala pozostávať z troch hlavných modulov, no ešte predtým bude potrebné prepojiť a spojazdniť databázu spolu s aplikáciou.

3.4.1 Modul licencie

Prvý a zároveň najdôležitejší modul bol licencie. V tomto module by ako prvé malo byť možné vyhľadávanie licencií pre prípadné nájdenie určitej licencie v zozname. Z dôvodu vyššej prehľadnosti budú dané licencie zobrazované iba v stanovených množstvách.

Úvodná stránka modulu licencií by taktiež mala obsahovať dáta súvisiace s danou licenciou, ako informácie o dodávateľovi, zodpovednej osobe avšak nebude sa jednať o kontaktnú osobu k dodávateľovi ale osobu priradenú k danej licencii a počet voľných licencií danej licencie. Pod voľnou licenciou sa rozumie licencia, ktorej exspirácia nebude mať k sebe priradeného žiadneho používateľa. Tento modul by mal obsahovať aj možnosť si zobraziť detailnejšie informácie jednotlivých licencií. Detailný pohľad by mal zobrazovať informácie o samotnej licencii rovnako ako jej dodávateľa, samotné meno, zodpovednú osobu z firmy a prípadne inštalačný odkaz k licencii. Pre potrebu udržiavania prehľadnosti v nákladoch na konkrétnu licenciu bude detail obsahovať aj finančnú štatistiku. To bude súhrn dát cien exspirácií ku konkrétnej licencii, teda kompletný súhrn cien exspirácií, priemernú cenu, najdrahšie a najlacnejšie obdobie licencie, k tomu aj počet voľných a využívaných licencií. Ďalšou vecou v detaile by mala byť možnosť mať prehľad o exspiráciách momentálnej licencie. Keďže sa exspirácie dajú rozdeliť do viacerých skupín, tak bude potreba vytvoriť možnosť si ich odfiltrovať. Z dôvodu archivovania vo firme pre prípadné vytvorenie hlásení nechceme žiadnu exspiráciu zmazať, preto budeme rozlišovať medzi aktívnou a neaktívnou exspiráciou. Budeme teda mať aktívne, neaktívne a voľné exspirácie. Tabuľka exspirácií by mala obsahovať dátum platnosti a dátum exspirácie, cenu danej exspirácie, používateľa, ktorý používa danú licenciu a hlavne či je exspirácia k licencii použiteľná aj po vypršaní dátumu platnosti.

Neprihlásený používateľ by mal mať prístup k tomuto modulu a k úvodnému pohľadu, no bude mu umožnené vyhľadávanie zadaných licencií a prípadné zobrazenie detailných informácií o konkrétnej licencii. Z bezpečnostných dôvodov by nemal mať možnosť pridať nový licenciu alebo upraviť už existujúcu konkrétnu licenciu. Zodpovedná osoba bude môcť pozmeniť údaje iba ku nej priradenej licencii. Tento používateľ bude taktiež mať možnosť vytvárať nové exspirácie a k nim neskôr prípadne priradiť alebo odobrať osobu, ktorá by využívala danú exspiráciu pre licenciu. Taktiež

bude môcť aktívnu exspiráciu urobiť neaktívnou. Manažér spolu s administrátorom bude mať rovnaké právomoci s tým rozdielom, že budú môcť tieto činnosti vykonávať pre všetky licencie spolu s možnosťou pridať novú licenciu do systému.

Z databázového pohľadu sa jedná tabuľky obsahujúce viacero atribútov rôznych dátových typov. Nad týmito tabuľkami je možné vykonávať CRUD operácie.

3.4.2 Modul dodávatelia

Ako druhý modul bolo potreba vytvoriť sprehľadnenie dodávateľov. Tento modul by mal na hlavnej stránke modulu obsahovať zoznam dodávateľov a k nim povrchové údaje. Pri detailnom zobrazení konkrétneho dodávateľa bude mať informácie o dodávateľovi ako jeho meno, IČO, samotný odkaz na portál dodávateľa a k nim prihlasovacie meno a prihlasovacie heslo, aby sa bolo možné prihlásiť k získaniu licencie. Ďalej by tu mali byť kontaktné osoby pre určitého dodávateľa. K týmto osobám bude uvedený email pre možnosť kontaktu spolu s telefónnym číslom.

Možnosť pridávať alebo upravovať aktuálnych dodávateľov by mal mať iba manažér alebo administrátor, nie však bežný zamestnanec.

3.4.2.1 Modul kontaktné osoby

Ďalší modul mal byť pre kontaktné osoby. Podobne ako oba predošle spomenuté moduly aj tento by mal mať možnosť vyhľadávať pre rýchlu potrebu nájsť konkrétnu kontaktnú osobu. Ďalej by mal obsahovať aj informáciu o dodávateľovi, ku ktorému je kontaktná osoba priradená s možnosťou presmerovania.

Neprihlásený používateľ by mal mať možnosť prístupu ku kontaktným osobám, avšak pridať kontaktnú osobu ku dodávateľom môže iba administrátor alebo manažér.

3.5 Tretie zadanie

Mojím tretím a posledným zadaním bolo vytvoriť modul pre export. Tento modul by mal umožňovať vyhľadávanie licencií, o ktorých si bude požívateľ môcť exportovať správu z časového rozsahu, ktoré si určí.

Avšak exportovať všetky správy si bude môcť iba manažér alebo administrátor, bežný používateľ si môže exportovať iba licencie, ku ktorým je pridelený ako zodpovedná osoba.

Kapitola 4

Použité technológie

4.1 MySQL

Štruktúrovaná kolekcia dát sa nazýva databáza. Pokiaľ sú dáta uložené v tabuľkách, ktoré uchovávajú konkrétne množiny dát tak sa jedná o relačnú databázu.

MySQL je otvorený relačný databázový systém (RDBMS) s modelom klient-server. RDBMS je softvér, ktorý nám slúži na vytváranie a správu databáz, ktoré sú založené na relačnom modeli. Pomocou MySQL sa vytvorí databáza, v ktorej budú dáta uložené a bude možné s nimi manipulovať. Klient potom spolu so serverom spolu komunikujú pomocou jazyka štruktúrovaného vyhľadávacieho jazyka SQL tým, že vytvárajú dotazy. Následne im server vráti požadovanú odpoveď, ktorá sa objaví na klientovej strane [1].

4.2 Docker

Docker ponúka možnosť uložiť a spustiť infraštruktúru danej aplikácie v takzvaných kontajnerov. Vďaka izolovanosti je možné spustiť viac kontajnerov naraz na danom hostiteľovi. Docker používa klient-server architektúru, kde používateľ komunikuje pomocou klienta s démonom, ktorý sa stará o beh, vytvorenie a distribuovanie kontajnerov. Kontajner je priestor, ktorý obsahuje všetko potrebné pre chod aplikácie. Pomocou týchto kontajnerov môžu aplikácie fungovať oddelene od zbytku systému [2].

4.3 Java EE

Java EE znamená Jave Enterprise Edition, predtým známa ako J2EE. Java EE sa používá napríklad v bankových informačných systémoch, elektronickýc obchodoch alebo účtovníctvách. V podstate ide o kolekciu špecifikácií obklopujúce prostredie Java SE (Standard Edition). Podnikové funkcie, ako

webové služby a distribuované výpočty sú súčasťou platformy pre vývojárov, ktorú ponúka Java EE.

Java EE má rôzne druhy špecifikácií, ktoré sú prospešné pri čítaní a zápisu dát z databáze transakčným spôsobom alebo vytváraní webových stránok. V jave EE nájdeme viaceré API tieto obsahujú funkcie základných rozhraní API Java SE medzi, ktoré patria servlety, konektrory, Enterprise JavaBeans, niekoľko technológií webových služieb a stránky serveru [3].

4.4 Spring Boot

Spring Boot je otvorený mikro framework, ktorý spravuje spoločnosť Pivotal. Tento framework poskytuje všetky funkcie springu, keďže sa jedná o nadstavbu. Server Tomcat nastaví a vytvorí server dostupný pre prehliadače pri spustení projektu [4].

4.5 Thymeleaf

Je to otvorený softvér vytvorený v roku 2011 Danielom Fernándezom. Je to moderný Javovský šablónovací systém pre web a prostredia schopné spracovať CSS, samotný text, JavaScript, XML a HTML. Thymeleaf je navrhnutý tak, aby bol schopný dodržať webové štandardy hlavne HTML5. Cieľom tohto šablónovacieho systému je poslúžiť udržateľným spôsobom vytvárania šablón [5].

4.6 JavaScript

JavaScript je objektovo orientovaný, interpretovaný jazyk obsahujúci prvotriedne funkcie hlavne využívaný na riadenie chovania webových stránok. Jazyk s prvotriednymi funkciami umožňuje predávanie funkcií ako argument ďalšej funkcii a ich vracanie ako hodnoty z druhých funkcií. JavaScript je najčastejšie používaný skriptovací jazyk pre webové stránky, avšak je používaný aj v iných oblastiach. Tento jazyk beží na klientovej strane na webu, túto vlastnosť je možné použiť k naprogramovaniu správania webovej stránky pri výskyte udalosti [6].

4.7 HTML

Skratka HTML značí hypertextový značkovací jazyk, používaný na vytváranie webových aplikácií a stránok. Pomocou tohto jazyka sa definuje štruktúra webu.

Html je sada tagov, ktoré hovoria prehliadaču ako zobraziť stránku. Tieto príkazy sa volajú tagy a používajú sa k označeniu, zabaleniu alebo ohraničeniu rôznych častí obsahu. Hypertextové odkazy umožňujú prepájanie webových stránok v rámci jednej alebo viacerých stránok [7].

4.8 CSS

Kaskádové štýly skrátene CSS je nenáročný dizajnový jazyk, ktorý chce uľahčiť dizajn webových stránok. Najbežnejšie sa CSS používa spolu s HTML. CSS slúži na upravovanie a spracovanie vzhľadu webovej stránky ako rozloženie stĺpcov, farby a štýl písma alebo rôzne formy zobrazenia pre rozdielne zariadenia a veľkosti obrazoviek [8].

4.9 SCSS

Sassy kaskádové štýly je preprocesor, ktorý umožňuje používať funkcie, ktoré sa ešte nenachádzajú v širšom štandarde CSS. Samotné SCSS rozširuje sytax CSS, každý štýl, ktorý sa dá použiť v CSS je teda možné použiť aj v SCSS a pritom sa jeho význam nezmení. Pomocou tohto preprocesoru sa zníži počet opakovaní a uľahčí sa údržba štýlov [9].

4.10 jQuery

Framework jQuery je multiplatformová, malá, rýchla JavaScriptová knižnica, ktorá má zjuednodušiť skriptovanie HTML na strane klienta. Umožňuje prechádzať a upravovať DOM elementy taktiež je používaný k pridávaniu animácií [10].

4.11 Bootstrap

Jedná sa o najpopulárnejší framework pre vývoj responzívnych webových aplikácií. Je to kolekcia už vopred napísaných znovu použiteľných častí kódu z HTML, JavaScriptu a CSS. Základná komponenta boostrapu je kontajner, v ktorom sú ukladané ostatné prvky stránky [11].

4.12 IntelliJ

InteliJ je otvorená platforma, ktorá je hlavne používaná na vývoj programov. Obsahuje mnoho pomocných nástrojov k uľahčeniu písania kódu. Toto prostredia má taktiež pokročilú kontrolu chýb a prípadné návrhy na ich opravu a mnoho iných asistenčných funkcií, ktoré uľahčujú používanie a vytváranie aplikácií [12].

Kapitola 5

Návrh a implementácia

5.1 Úložisko dát

Keďže oddelenie, v ktorom som prax vykonával používa ako úložisko výlučne MySQL, tak som databázu musel vytvoriť v MySQL. Tento databázový systém nám ponúka pri tvorení databáze si vybrať rôzny formát úložiska dát takzvaný storage engine. Osobne som zvolil najnovšiu verziu s formátom úložiska InnoDB pre prácu s tabuľkami. Tento typ úložiska je taktiež predvolený od verzie 5.5 a vyššie. Vo firme a v oddelení je mnoho ľudí, takže k systému bude pristupovať viac používateľov. Dátové úložisko InnoDB pomocou konzistentného odblokovania čítania zvyšuje výkon pokiaľ je databáza používaná vo viac užívateľskom prostredí. Úplný diagram databáze nájdeme v prílohe A. Nižšie popísané tabuľky je možné prepojiť pomocou cudzích kľúčov, a to pomocou zvoleného datového úložiska InnoDB.

Samotná databáza pozostáva zo siedmych tabuliek a to *supplier*, *contact person*, `m_user`, *group*, *group member*, *license* a *expiration*.

Tabuľka `m_user` čiže používateľ sa dá považovať za najrozsiahlejšiu, keďže každý zamestnanec firmy má vo firme svoje užívateľské meno, heslo na prihlásenie sa k svojmu firemnému účtu a firmou pridelený email. Heslo bude z dôvodu väčšej bezpečnosti ukladané v databáze v zašifrovaneju forme. V tejto firme pracujú zamestnanci z viacerých krajín, tak preto je potrebné, aby obsahovala kontaktné údaje o zamestnancovi. Ako v každej spoločnosti majú zamestnanci svoju pozíciu v oddelení alebo už v tíme, preto sa bude evidovať aj pozícia zamestnanca. Najdôležitejšia pozícia z pohľadu systému je manažér.

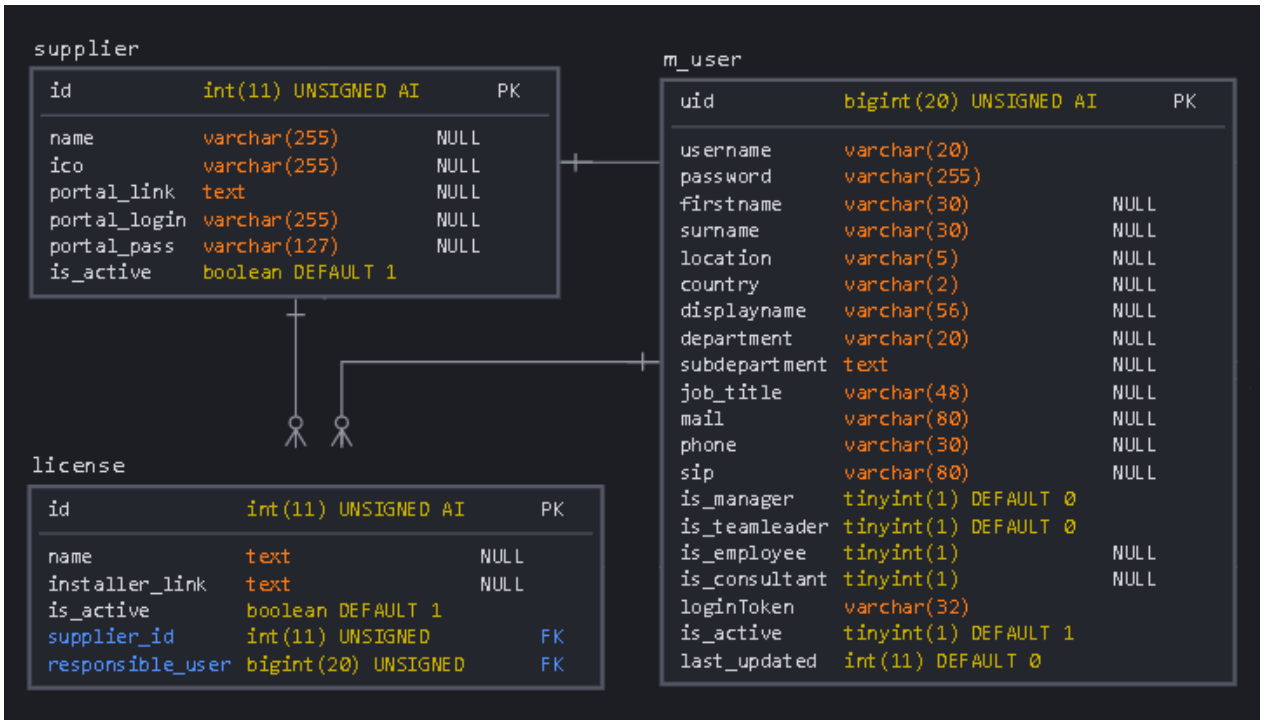
Ďalej existuje tabuľka pre licencie *license*, ktorá obsahuje textové polia, v ktorých sa ukladajú informácie o licencii, ďalej sú tu cudzie kľúče k dodávateľovi a zodpovednej osobe. Vzťah popisujúci väzbu medzi licenciou a dodávateľom, licenciou a zodpovednou osobou, popisuje stav aký je vo firme, keďže licencia má jednu zodpovednú osobu a môže mať iba jedného dodávateľa.

Medzi ďalší dôležitý prvok databáze patrí tabuľka *supplier*, ktorý udržiava informácie o dodávateľoch a obsahuje prevažne textové polia.

V tabuľke *expiration* sa udržiava dátum platnosti licencie až do jej vypršania v unixovom dátovom formáte, keďže sa jedná o číselný formát táto tabuľka obsahuje prevažne číselné a booleovské hodnoty pre prípad, kedy by bola licencia použiteľná aj v prípade uplynutia dátumu exspirácie.

Tabuľka ***m_group*** slúži na rozdelenie zamestnancov na základe pozícií zamestnancov vo firme. Pomocou tejto tabuľky sa umožnia prípadne zakážu používateľom niektoré funkcie v rámci aplikácie.

Tabuľka ***contact_person*** obsahuje informácie vo forme textových polí pre kontaktnú osobu, spolu s popisom zodpovednosti. Tabuľka ***contact_person*** je prepojená s tabuľkou *supplier*.



Obr. 5.1: Diagram vzťahu medzi licenciou a dodávateľom a licenciou a užívateľom

V databáze platia tieto pravidlá:

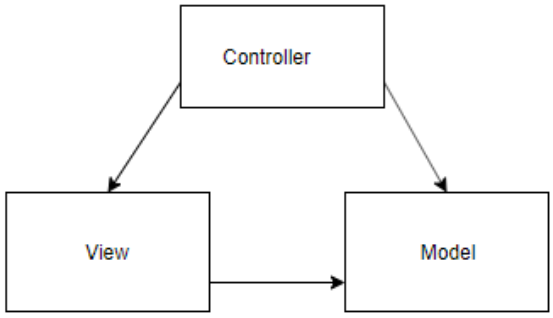
- Zodpovedná osoba spolu s dodávateľom, môže mať na starosť viacero licencií, avšak licencia môže mať iba jednu zodpovednú osobu a jedného dodávateľa, teda platí relácia 1:N (one-to-many)
- Ku každej licencii môže existovať viac exspirácií, no každá exspirácia môže mať len jednu licenciu, takže tu je relácia 1:N (one-to-many)
- Kontaktná osoba má jedného dodávateľa, avšak môže nastať situácia, kedy budú k dodávateľovi pridelené viaceré kontaktné osoby relácia 1:N (one-to-many)
- Užívateľ môže byť vo viacerých skupinách a podobne jedna skupina môže obsahovať viac užívateľov. Táto relácia sa volá M:N (many-to-many)

```
CREATE TABLE IF NOT EXISTS m_user
(
    uid bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT,
    username varchar(20) NOT NULL,
    password varchar(255) NOT NULL,
    firstname varchar(30),
    surname varchar(30),
    location varchar(5),
    country varchar(2),
    displayname varchar(56),
    department varchar(20),
    subdepartment text,
    job_title varchar(48),
    mail varchar(80),
    phone varchar(30),
    sip varchar(80),
    is_manager tinyint(1) DEFAULT 0 NOT NULL,
    is_teamleader tinyint(1) DEFAULT 0 NOT NULL,
    is_employee tinyint(1),
    is_consultant tinyint(1),
    loginToken varchar(32),
    is_active tinyint(1) DEFAULT 1 NOT NULL,
    last_updated int(11) DEFAULT 0 NOT NULL,
    PRIMARY KEY (`uid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Výpis 5.1: Zdrojový kód tabuľky `m_user`

5.2 Architektúra

Firma rozhodla aby sa použila architektúra MVC pre štruktúru aplikácie, hlavne z dôvodu zrozumiteľnosti a ľahšej udržateľnosti celkového kódu, ktorú poskytuje MVC tým, že rozdelí aplikáciu na tri časti. Tieto časti sú navzájom prepojené a vzájomne závislé. Jedná sa o vhodnú a obľúbenú voľbu, pretože tento architektonický model sa pomerne mnoho používa pri vytváraní webových aplikácií. Model-View-Controller alebo MVC rozdeľuje aplikáciu na tieto časti.



Obr. 5.2: Vzťahy medzi časťami MVC architektúry

- Model je komponent, ktorý pozostáva z tried a tieto triedy majú na starosť celkovú prácu s dátami. Patria sem spojenia s databázou, všetky databázové dotazy a akákoľvek logika vykonávaná s dátami. Ostatné časti aplikácie pristupujú k dátam iba cez model.
- View (Pohľad) Tým, že priamo komunikuje s užívateľom mu pomáha zobraziť údaje o modeli.
- Controller (Ovládač) zabezpečuje a spracováva interakciu používateľa. Podľa interakcií používateľa ovládač mení dáta v modelu alebo aktualizuje samotné pohľady [13].

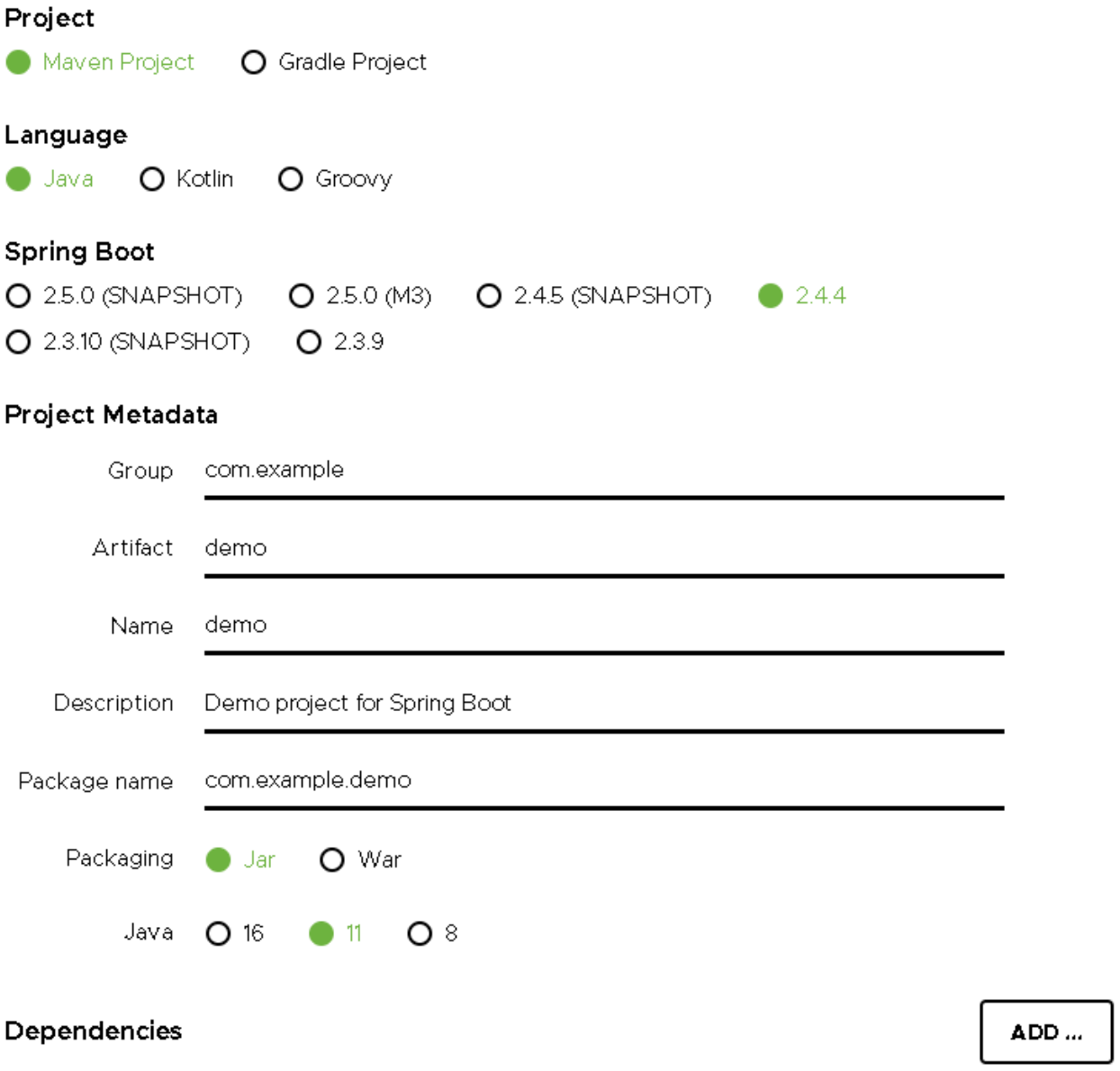
5.3 Spring initializr

Pomocou tejto webovej aplikácie môžeme vygenerovať prvotnú štruktúru Spring Boot projektu. Tento webová aplikácia avšak nevygeneruje žiaden aplikačný kód, ale vytvorí iba základnú projektovú štruktúru a buď Maven alebo Gradle špecifikáciu zostavenia, pomocou ktorej môžeme zostaviť svoj kód. Vďaka tejto službe už nebude potrebné vytvárať štruktúru ale iba zdrojový kód aplikácie.

Na začiatku je na výber medzi vyššie spomínaním Mavenom a Gradle. Oba systémy slúžia na zostavovanie aplikácií, ktoré sú vyvíjané na platforme Java. Rozhodol som sa pre tento projekt použiť Maven. Výhodou použitia tohto systému je, že sa zbavíme použitia závislosti na určitom vývojovom prostredí. Keďže všetky informácie, ktoré potrebuje na zostavenie a skompilovanie projektu má rovno uložené v pom.xml jedná sa o špeciálny súbor POM.

Ako ďalšie sú na výber programovacie jazyky a verzia Spring Boot frameworku. V čase vytvárania projektu bola dostupná stabilná verzia 2.2.2, ktorú som následne zvolil spolu s firmou zadaným jazykom Java spolu s jej ôsmou verziou. Ďalej je požadované vyplnenie základných informácií o

vytváranom projekte, to zahŕňa meno projektu, meno balíčka, skupinu, meno spolu s artefaktom a samotný popis.



Obr. 5.3: Webové rozhranie spring initializr

Ďalšia hlavná funkcia systému Maven je riešenie závislostí. Pomocou tejto funkcie nebudem musieť ručne kopírovať knižnice a lokalizovať ich na classpath. Toto je užitočná funkcionalita, ktorá sprehľadní a zjednoduší vývoj pri použití veľkého množstva závislostí [14].

Posledná časť je špecifikovanie závislostí. Závislosť sú súbory alebo programy, ktoré sú potrebné pre funkčnosť hlavného programu.

Použité závislosti:

- spring-boot-starter-web - Tento štartér pre Spring web používa ako predvolený zabudovaný server Tomcat, REST a Spring MVC. Táto závislosť tiež znižuje počet závislostí, ktoré je potreba zadať ručne a to tým, že prechodne priťahuje všetky závislosti, ktoré súvisia s vývojom webu [15].

- spring-boot-starter-data-jpa - Táto závislosť zabezpečuje efektívne spojenie s Spring aplikácie s relačnou databázou. Vnútorne využíva závislosť spring-boot-jpa, ktorá využíva funkcie definované v JPA špecifikácii. S pridaním vlastných funkcií ako vytváranie databázových dotazov z názvu metódy a bezkódová implementácia modelu úložiska bez kódu [16].

- mysql-connector-java - Jedná sa o MySQL ovládač pre podporu prepojenia aplikácie s databázou MySQL [17].

- spring-boot-starter-thymeleaf - Implementuje modul šablonovacieho systému Thymeleaf. Spring Boot následne nakonfiguruje šablónovací systém, aby čítal súbory šablón z /resource/templates [18].

- spring-boot-starter-security - Závislosť, ktorá sa používa pre zabezpečenie Spring Boot aplikácie [19].

- thymeleaf-extras-springsecurity5 - Pridá podporu Thymeleafu pre Spring Security.

- spring-boot-starter-mail - Vďaka tejto závislosti je možné odosielať e-maily Java Mail a taktiež posielanie e-mailov cez Spring Framework [20].

- spring-boot-devtools - Tento modul poskytuje nástroje, ktoré slúžia na spríjemnenie vývoja počas práce na Spring Boot aplikácii [21].

- spring-boot-starter-test - Pre vytváranie jednoduchých testov na overenie funkčnosti je potrebné pridať túto závislosť

- super-csv - Modul, ktorý umožní používanie Javovskej knižnice pre CSV. Bude využívaný pre generovanie výkazov pre oddelenie [22].

5.4 Anotácie

Anotácie v Spring Boot sú metadáta vo forme, ktoré poskytujú údaje o programe. Teda anotácie sú použiteľné na dodávanie doplňujúcich informácii o programe, avšak nie sú súčasťou aplikácie, ktorú vyvýjame. Anotácie priamo neovplyvňujú kód, ktorý označujú.

V aplikácii som označil hlavnú triedu pomocou anotácie *@SpringBootApplication*, ktorá sa používa k povoleniu funkcií ako napríklad nastavenie automatického vyhľadávania REST kontrolérov a ich registráciu v Springu. Ďalej umožňuje konfiguráciu založenú na Jave a auto konfiguračnú

funkcionalitu. V staršej verzii Springu bolo potrebné hlavnú triedu označiť viacerými anotáciami *@Configuration*, *@ComponentScan*, *@EnableAutoConfiguration* pre dosiahnutie tejto funkcionality. V súčasnosti anotácia *@SpringBootApplication* nahrádza tieto anotácie.

5.5 Modely

Všetky tabuľky v databáze majú svoje atribúty. Aby sa tieto vlastnosti dali presunúť do aplikácie a následne s nimi pracovať je potrebné k týmto tabuľkám vytvoriť ich reprezentáciu v aplikácii pomocou modelových tried.

Tieto tabuľky obsahujú určité spoločné vlastnosti, ktoré by sa dali špecifikovať v jednej triede. Ostatné triedy by následne z tejto triedy dedili dané spoločné vlastnosti, aby sa nemuseli opakovane vypisovať do každej triedy zvlášť. Z tohto dôvodu som vytvoril jednu triedu, ktorá tieto vlastnosti zahŕňa. Takáto trieda musí byť v Spring Boot označená špeciálnou anotáciou *@MappedSuperclass*. Táto anotácia povoľuje ostatným entitám dediť vlastnosti z triedy, ktorá je takouto anotáciou označená.

Najhlavnejšia zložka každej tabuľky je jej identifikátor. *@Id* je špeciálna anotácia používaná na označovanie primárnych kľúčov, ktorá je nezávislá na implementácii. V štruktúre vytvorenej databáze som zvolil automaticky generovaný identifikátor záznamu. Túto vlastnosť je v modelových triedach možné napodobniť pomocou Springovskej anotácie *@GeneratedValue*, ktorá ponúka štyri typy generovania [23]

- AUTO
- IDENTITY
- SEQUENCE
- TABLE

Nastavil som stratégiu generovania pre Identity, pretože tento generátor očakáva hodnoty vytvorené pomocou identifikátora v databáze a teda sa automaticky zvyšujú.

Premenné, ktoré budú reprezentovať atribúty z tabuliek podľa ich dátových typov je potrebné označiť anotáciou *@Column*. Pre lepšiu prehľadnosť v programovom kóde som využil užitočnú funkcionalitu, ktorú táto anotácia ponúka teda je im možné prideliť vlastné pomenovanie. Triedne premenné, v ktorých som nevyužil funkcionalitu tejto anotácie použijú predvolenú hodnotu, teda meno premennej.

Pre atribúty, ktoré vyžadovali podľa návrhu databáze byť nenulové som označil pomocou anotácie *@NotNull*. Táto anotácia nepovolí, aby daná vlastnosť mala nulovú hodnotu.

Ostatné modelové triedy, z ktorých sa nebudú ďalej preberať spoločné vlastnosti, som označil pomocou anotácie *@Entity*. Trieda, ktorá je označená takouto anotáciou určuje, že trieda je entita a

je namapovaná na existujúcu tabuľku v databáze. Ďalšia dôležitá anotácia, ktorú som použil hneď po *@Entity* je *@Table*, ktorá umožňuje určiť podrobnosti tabuľky. Použil som vlastnosť anotácie *@Table*, ktorá umožňuje určiť názov databázovej tabuľky, ktorá sa má použiť na mapovanie. Pokiaľ by som nepoužil túto anotáciu, tak by sa použilo triedne meno, čo by znižovalo prehľadnosť kódu. Opäť som využil užitočnú, avšak inú funkcionalitu anotácie *@Column*. Tentokrát pri tvorení modelu pre používateľa, aby som mu stanovil maximálnu veľkosť dĺžky hesla pomocou vlastnosti length. V samotnom modeli používateľa som ešte niekoľkokrát využil funkcionalitu *@Column* pre stanovenie vlastného pomenovania.

```
@OneToMany(mappedBy = "license")
 @OrderBy("validFrom")
 private Set<Expiration> expiration;
```

Výpis 5.2: Strana 1 relácie 1:N licencií a exspirácií

```
@ManyToOne
 @JoinColumn(name = "license_id")
 private License license;
```

Výpis 5.3: Strana N relácie 1:N licencií a exspirácií

Pre prepojenie tabuliek a vytvorenie relácie one-to-many používateľa s exspiráciami a licenciami, ktoré sú mu priradené ako zodpovednej osobe, som v používateľovej modelovej triede označil triednu vlastnosť anotáciou *@OneToMany*, ktorá definuje vzťah medzi dvoma entitami [24]. Toto prepojenie je spojené s modelovou triedou, ktorá bude reprezentovať druhú stranu. Keďže exspirácií a licencií môže mať jeden používateľ viac, tak som musel označiť danú vlastnosť modelovej triedy pomocou anotácie *@ManyToOne* spolu s *@JoinColumn*, ktorý špecifikuje stĺpec cudzieho kľúča. Pokiaľ by som nepoužil anotáciu *@OrderBy* použila by sa predvolená forma zoraďovania [25]. Táto anotácia funguje iba pokiaľ sa použije s relačnými anotáciami *@OneToMany* alebo *@ManyToMany* [26].

```
@ManyToMany
 @JoinTable(name = "group_member",
     joinColumns = { @JoinColumn(name = "user_id") },
     inverseJoinColumns = { @JoinColumn(name = "group_id") })
 private Set<Group> groups;
```

Výpis 5.4: Strana M relácie M:N používateľov a skupín

```
@ManyToMany(mappedBy = "groups")
 private Set<User> users;
```

Výpis 5.5: Strana N relácie M:N používateľov a skupín

Ďalším z dôležitých vzťahov je vzťah medzi používateľom a skupinou na základe, ktorého sa budú používatelia aplikácie rozdeľovať do skupín, ktoré im povolia, prípadne zakážu niektoré funkcie v aplikácii. Tento vzťah sa dá definovať ako relácia many-to-many, keďže niektorý používatelia môžu patriť do viacerých skupín a naopak. K vytvoreniu takéhoto vzťahu som využil anotáciu *@ManyToMany*, ktorá som použil v oboch modelových triedach spolu s *@JoinTable*, ktorá definuje spojenie dvoch pridružených entít [27]. Využil som aj anotáciu *@Transient* , ktorá sa používa na označenie triednej premennej, ktorá sa nemá udržiavať v databáze [28]. Jedná sa o premennú typu boolean, ktorá určuje či sa jedná o administrátora alebo nie. Pomocou rovnakých anotácií som vytvoril zostávajúce modelové triedy.

5.6 Repozitáre

Pre uchovávanie SQL dotazov som vytvoril rozhrania, pre každú modelovú triedu kvôli prehľadnosti v zdrojovom kóde. Na vytvorenie dotazu SQL som použil anotáciu *@Query*, do ktorej som napísal vlastný dotaz a stanovil som mu návratovú hodnotu. V niektorých prípadoch bolo potrebné pridať parametre, ktoré som pridal pomocou anotácie *@Param*. Takýmto spôsobom som vytvoril potrebné dotazy pre ostatné

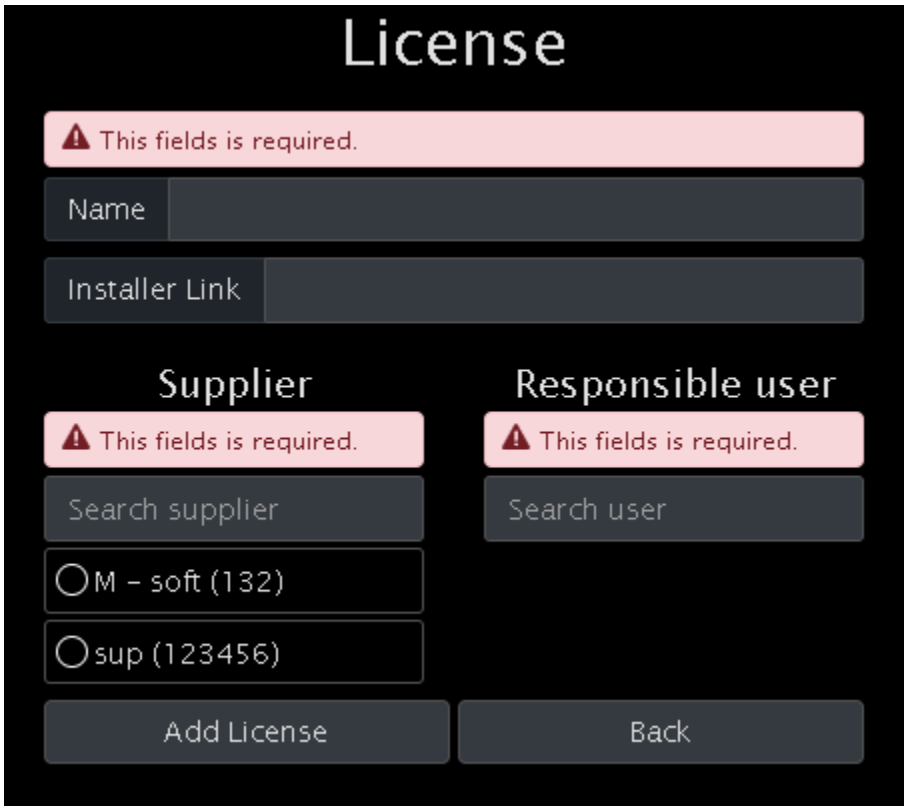
```
@Query("SELECT license FROM License license WHERE license.name LIKE %:name% AND
    is_active = 1")
 Page<License> findByNameActive(@Param("name") String name, Pageable pageable);
```

Výpis 5.6: Databázový dotaz v licenčnom repozitáry

5.7 Validátori

Rozhodol som sa vytvoriť validáciu uživateľských vstupov aj na strane serveru z dôvodu vyššej bezpečnosti, keďže validácia ktorá prebieha na strane klienta pomocou Javascriptu môže byť obídená. Spring obsahuje rozhranie Validator, ktoré sa môže použiť na validáciu objektov. Rozhranie Validator funguje pomocou objektu Errors, takže počas overovania môžu validátori hlásiť zlyhanie overenia objektu Errors.

Takéto validačné triedy budú implementovať vyššie spomínané rozhranie. Metódy v týchto triedach som označil anotáciou *@Override*. Táto anotácia poukazuje, že metóda podradenej triedy prepisuje svoju metódu základnej triedy. V tejto validačnej triede sú dve metódy. Jedna slúži na to, aby sa určilo či daný validátor môže overiť inštancie dodanej triedy a druhá vykonáva samotné overovanie konkrétnej inštancie, ktorá avšak musí byť podporovaná.



Obr. 5.4: Upozornenie v prípade nesprávneho vyplnenia formy

Najkratšie a najjednoduchšiu metódu validate obsahuje trieda SupplierValidator spolu s triedou LicenseValidator, v ktorých sa overuje jednoduché premenné. Rozsiahlejší obsah sa nachádza vo validačnej metóde pre kontaktnú osobu teda v triede ContactPersonValidator. V tejto triede som vytvoril regulárne výrazy na overenie emailovej adresy spolu s telefónnym číslom, ktoré používateľ zadá pri prípadnom vytváraní alebo úprave už existujúcej kontaktnej osoby.

5.8 Prihlasovanie používateľa

Začal som vytvorením dvoch tried, ktoré budú slúžiť k prihlasovaniu používateľov do aplikácie. Jedna z týchto tried CustomUserDetails implementuje rozhranie UserDetails. Táto trieda bude mať v sebe uložené informácie o používateľovi, ktoré sú potrebné pre prihlasovanie [29].

Druhá trieda CustomUserDetailsService implementuje rozhranie UserDetailsService. Táto trieda slúži na načítanie načítanie dát o používateľovi, ktorej metóda nájde podľa prihlasovacieho mena v databáze pokiaľ existuje a tohto používateľa predá triede CustomUserDetails. Taktiež som použil anotáciu *@Autowired* na repozitárovej inštančnej premennej. Pomocou tejto anotácie Spring deteguje, že daný objekt je potrebné nastaviť pomocou dependency injection [30].

Ďalšou dôležitou triedou, ktorú som vytvoril je WebSecurityConfig, ktorá rozširuje triedu WebSecurityConfigurerAdapter. Táto trieda sa zaoberá konfiguráciou bezpečnosti tejto webovej aplikácie. Túto triedu som označil s anotáciami *@Configuration* a *@EnableWebSecurity*. Anotácia *@Configuration* naznačuje, že takto anotovaná trieda má v sebe metódy obsahujúce anotáciu *@Bean*, ktoré sa spracujú v Spring kontajneri a vygenerujú sa Spring beany, ktoré sa následne použijú v aplikácii [31]. *@EnableWebSecurity* povolí podporu zabezpečenia webu. Keďže heslá sú v databáze uložené v zašifrovanej forme z bezpečnostného dôvodu pomocou BCryptPasswordEncoder, ktorý poskytuje Spring Security, tak som vytvoril metódu na odšifrovanie tohto hesla [32]. V tejto triede som ešte vytvoril metódy, ktoré pracujú s používateľom a ďalej poskytujú autentifikáciu a označil ich anotáciou *@Bean* a teda tieto metódy produkujú bean, ktorý bude spravovaný Springovským kontajnerom. Ďalej som v tejto triede vytvoril metódu configure, v ktorej som nastavil presmerovanie po úspešnom, prípadne neúspešnom prihlásení alebo odhlásení používateľa.

5.9 Kontroléri

Kontroléri slúžia na spracovanie požiadavkou od používateľa. Spring ponúka anotáciu @Controller, ktorá označuje, že trieda anotovaná takouto anotáciou je kontrolór. Všetky triedy, ktoré plnia úlohu kontroléra som označil Springovskou anotáciou @Controller [33].

5.9.1 Hlavný kontrolér

Na spracovávanie požiadavkou na hlavnej stránke som vytvoril triedu MainController. V tejto triede som vytvoril metódu, ktorá mapuje domovskú URL adresu a vracia model hlavnej stránky. Aby táto metóda správne fungovala je potrebné ju označiť anotáciou @RequestMapping, ktorá mapuje konkrétne požiadavky HTTP na označenú metódu kontroléra [34].

5.9.2 Ostatné kontroléri

Ako prvé som v kontroléri zaviedol validátor tak, že som vytvoril metódu, ktorá registruje predtým vytvorený validátor pre daný kontrolér. Túto metódu som označil pomocou anotácie *@InitBinder*. Metóda, ktorá je označená takouto anotáciou sa bude vykonávať pri každom HTTP požiadavku pokiaľ v anotácii nešpecifikujeme hodnotu [35].

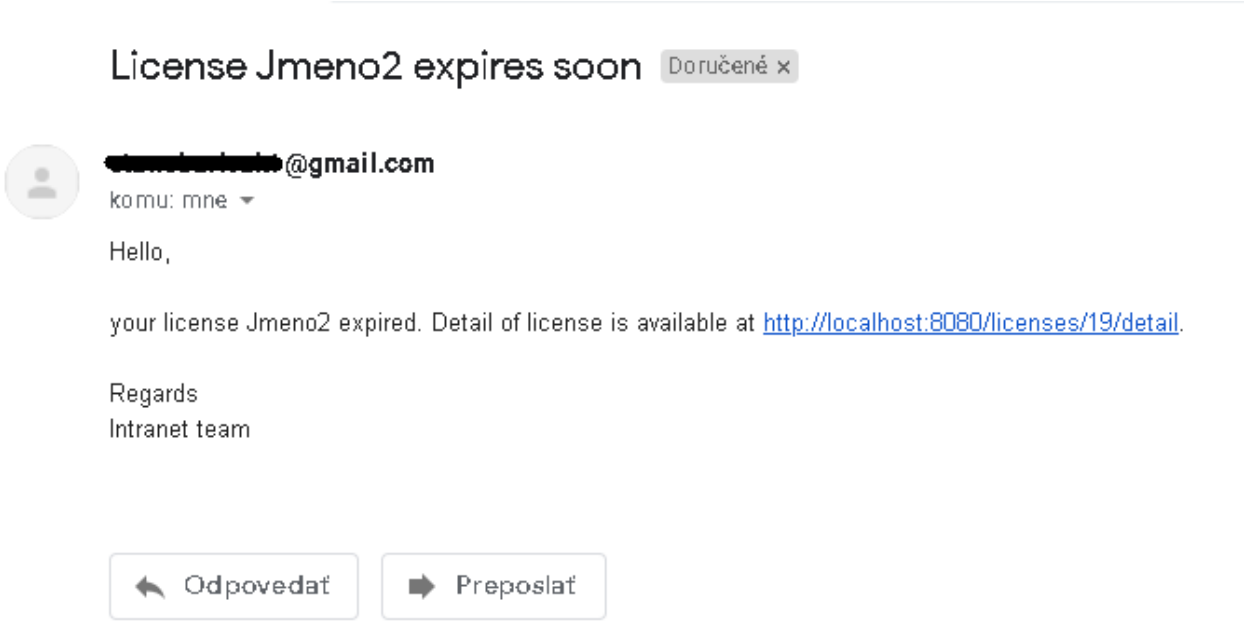
Ďalej som vytvoril metódu, ktorá bude obsluhovať URL požiadavku pre zobrazenie existujúcich licencií. Taktiež som do tejto metódy pridal nepovinné vstupné parametre, ktoré slúžia na zoraďovanie spolu so stránkovacím parametrom. Tieto vstupné parametre som označil anotáciou *@RequestParam* spolu s jej nastavením jej vlastností, ktorá sa používa na čítanie údajov a ich automatické naviazanie na parameter prítomný v poskytnutej metóde [36]. Avšak vstupný parameter pre stránkovanie som označil anotáciou *@PathVariable* vďaka, ktorej som extrahoval šablónovú časť URI predstavovanú premennou {page} [37]. Teda používateľ si bude môcť zoradiť informácie o li-

cenciách podľa mena, zodpovednej osoby, dodávateľa a počtu voľných licencií. Počet licencií, ktoré sa zobrazia na jednej stránke som nastavil na desať.

Ďalej si táto metóda získava informácie o prihlásenom používateľovi pomocou metódy, ktorá tohto používateľa vracia a následne povoľuje vytváranie nových licencií iba pre administrátora alebo manažéra.

Potom som vytvoril metódu, ktorá bude inicializovať novú licenciu a označil som ju anotáciou *@GetMapping*, ktorá mapuje HTTP požiadavky typu GET [38]. Následne druhá metóda už spracováva používateľom zadané dáta, ktoré sa ešte overia licenčným validátorom a pokiaľ všetko prebehlo úspešne, tak novú licenciu uloží a presmeruje pohľad k jej detailu, kde sa zobrazia podrobné informácie o samotnej licencii. Túto metódu som označil anotáciou *@PostMapping*. Takto označená metóda spracováva HTTP požiadavky typu POST [39].

Ďalej som vytvoril dve metódy, z ktorej jedna slúži na načítanie údajov existujúcej licencie do formulára pre prípadnú úpravu údajov tejto licencie. Druhá metóda slúži na spracovanie upravených údajov o licencii a ich následnom uložení. Obe metódy som označil potrebnými anotáciami na základe typu HTTP požiadavku. Taktiež som do týchto dvoch metód zaviedol overenie práv prihláseného používateľa.



Obr. 5.5: Emailové upozornenie o vypršanej exspirácii

Následne som vytvoril metódu, v ktorej som implementoval logiku detailného zobrazenia konkrétnej licencie. Spolu s informáciami o licencii sa zobrazia aj k nej príslušné exspirácie. Takýmto spôsobom som v každom kontroléri vytvoril metódy tohto typu. V exspiračnom kontroléri som ešte

vytvoril metódu, ktorú som označil anotáciou *@Scheduled* a nastavil vlastnosť tejto anotácie cron na spúšťanie tejto metódy každé štyri hodiny [40]. Táto metóda si prejde zoznam exspirácií ku danej licencii a v prípade, že niektorej z exspirácii sa blíži dátum konca platnosti za 7 dní pošle upozornenie spolu s odkazom na detail danej licencie na emailovú adresu zodpovednej osobe k nej priradenej. Pokiaľ už daná exspirácia vypršala rovnako sa tejto osobe pošle upozornenie na jej emailovú adresu.

5.10 Pohľady

Všetky pohľady som umiestňoval do predvoleného priečinka kde Spring Boot tieto súbory hľadá. Najskôr som si vytvoril šablóny, ktoré som neskôr používal pri vytváraní pohľadov pre jednotlivé moduly aplikácie.

Ako prvú som si vytvoril šablónu, ktorá tvorí telo každého pohľadu spolu s hlavičkou, ktorá obsahuje prihlasovacie okienko, ktoré sa ukáže pokiaľ naň klikne používateľ. Potom som sa sústredil na šablóny, v ktoré som neskôr používal pri vytváraní formulárov. Tieto šablóna boli veľmi užitočné, keďže jedna obsahovala vzor pre zaškrtávacie políčka a druhá obsahovala vzory pre textové, číselné alebo textové oblasti, takže pri vytváraní formulárov som už len zadal typ a pomenovanie jednotlivých častí tohto formulára. Dôležitou šablónou bola šablóna na stránkovanie, ktorú využívajú pohľady so zobrazovaním licencií, exspirácií alebo dodávateľov. Pre ušetrenie písania opakujúceho sa kódu pre ikony nodulov som taktiež vytvoril šablónu.

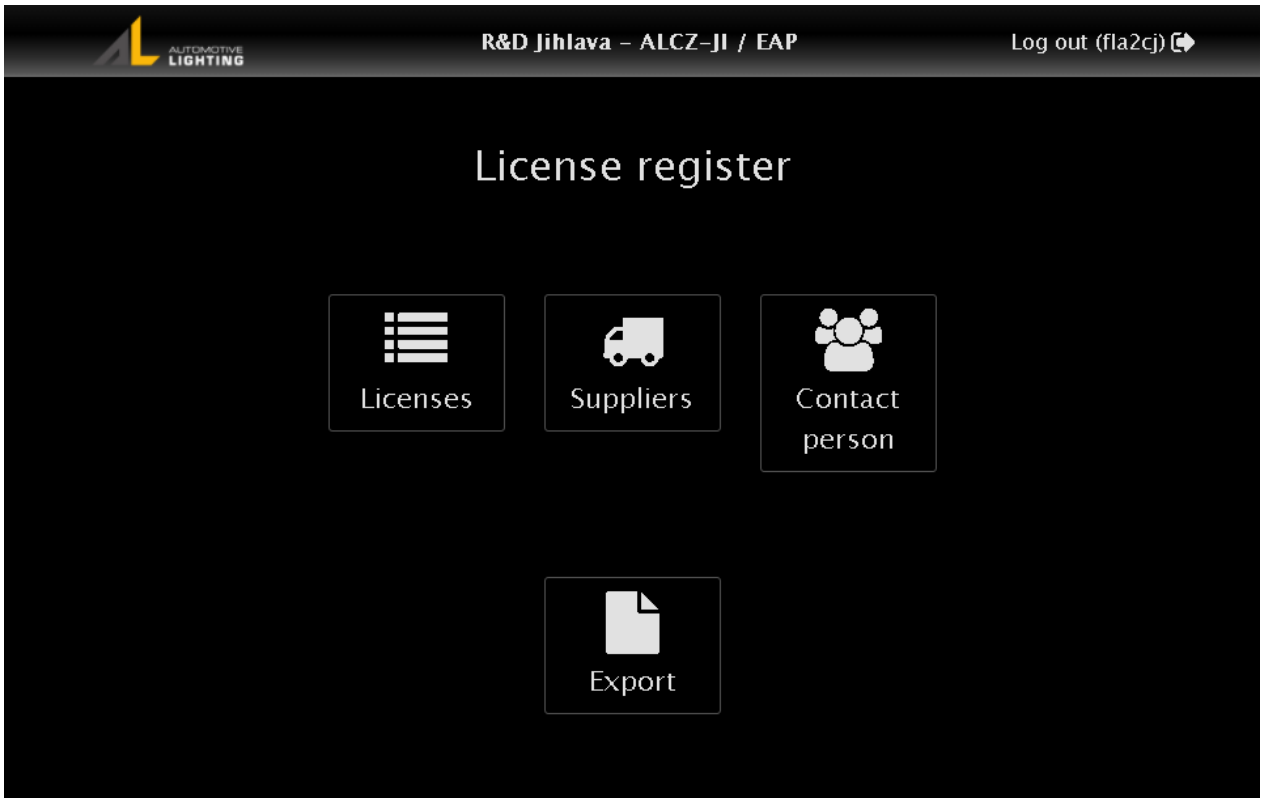
```
<html xmlns:th="https://www.thymeleaf.org">
 <body>

 <th:block th:fragment="link (label, link, icon)">
     <div class="lic-card col-3">
         <a class="lic-card__link" th:href="@{'/' + ${link}}">
             <div th:class="'lic-card__icon fa fa-' + ${icon} "></div>
             <div class="lic-card__label" th:text="${label}"></div>
         </a>
     </div>
 </th:block>

 </body>
 </html>
```

Výpis 5.7: Šablóna pre vytvorenie ikon modulov

Všetky formuláre sú okrem validácie na strane serveru taktiež validované na strane klienta pomocou JavaScriptu.



Obr. 5.6: Hlavná stránka aplikácie

5.11 Modul správ

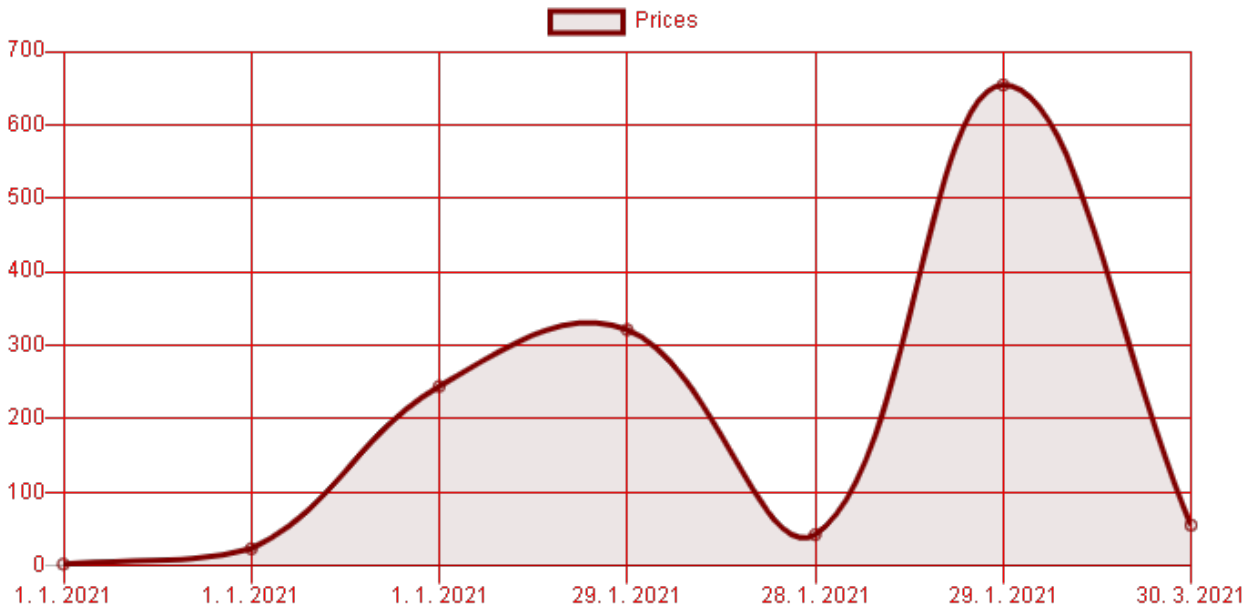
Keďže správa, ktorú si prihlásený používateľ bude môcť stiahnuť vo formáte CSV alebo v grafickej podobe. Ako prvé som teda do súboru pom.xml závislosť, ktorá umožní používať Javovskú knižnicu pre CSV. Následne som vytvoril triedu, ktorá obsahuje údaje ktoré sa budú exportovať.

Ďalej som sa sústredil na vytvorenie metód v kontroléri. Obe metódy som označil anotáciou @GetMapping. Taktiež prijíma tri nepovinné parametre označené anotáciou @RequestParam a to na vyhľadávanie a zadanie dátumov od kedy do kedy. Ďalej táto metóda spracováva dáta, ktoré jej zadal používateľ cez tieto parametre, ktoré potom predá druhej metóde a taktiež overuje právomoc prihláseného používateľa. Druhá metóda už zabezpečuje vygenerovanie súboru CSV.

```
<script th:inline="javascript">
    var prices = /*[[${prices}]]*/ [];
    var dates = /*[[${dates}]]*/ [];
 </script>
```

Výpis 5.8: Poslanie dát pre graf na ďalšie spracovanie

Samotný graf je generovaný pomocou JavaScriptovej knižnice Chart.js. Údaje na vygenerovanie grafu sú predávané z prvej metódy pomocou Thymeleafu.



Obr. 5.7: Graf z modulu export

Kapitola 6

Zhodnotenie a prínosy

6.1 Časová náročnosť

Časová náročnosť vykonávanej odbornej praxe bola zvládnuteľná rozložením úloh popri prezenčnom štúdiu vďaka voľným piatkom a štvrtkom.

Časovo najnáročnejšia časť vývoja bola vytvorenie funkcionality systému. Avšak značnú časový úsek zabralo odhaľovanie a následná oprava vyskytujúcich sa chýb. Naopak najmenej času pri vývoji spotrebovalo vytvorenie modelov mapujúcich tabuľky v databáze.

Prvok	Odhadovaná náročnosť (dni)	Skutočná náročnosť (dni)
Databáza	3	3
Modely	1	2
Kontroléri	10	15
Validácia	4	6
Prihlasovanie	3	5
Pohľady	5	7
Oprava chýb	x	13

Tabuľka 6.1: Časová náročnosť zadaných úloh

6.2 Uplatnenie znalostí získaných počas štúdia

V tejto časti budem popisovať znalosti nadobudnuté počas štúdia na VŠB, ktoré mi boli užitočné v priebehu vykonávania odbornej praxe.

Veľmi užitočné boli znalosti získané v predmetoch Úvod do databázových systémov a Databázové a informačné systémy. Na začiatku som ich využil hlavne pri návrhu a implementácii samotnej databáze, vrátane správneho použitia vzťahov medzi jednotlivými tabuľkami. Neskôr pri vytváraní SQL dotazov pomocou, ktorých si aplikácia získavala dáta z databáze.

Dôležité boli znalosti v oblasti objektovo orientovaného programovania, ktoré som nadobudol v predmete Programovanie II. Vďaka predmetu Programovacie jazyky 1 som mal znalosti s programovacím jazykom Java, ktoré som hojne využil. Na vyriešenie algoritmických problémov som využil znalosti z predmetov Algoritmy I a Algoritmy II.

Pri vytváraní pohľadov teda vizuálnej časti tejto webovej aplikácie mi boli veľmi užitočné znalosti nadobudnuté v predmete Vývoj internetových aplikácií. Predmet bol postavený tak, aby sme po dokončení mali taktiež prehľad o moderných trendoch v oblasti dizajnu webu. Užitočné boli hlavne jazyky HTML a CSS, ktoré tvorili základ každého pohľadu. Pri vytváraní funkcionality a overovaní vstupov formulárov som využíval skriptovací jazyk JavaScript, ktorý som sa taktiež naučil v tomto predmete.

Princípy správneho postupu, ktorý sa používa pri vytváraní systémov mi boli ukázané v predmetoch Úvod do softvérového inžinierstva a Vývoj informačných systémov.

6.3 Chýbajúce znalosti počas odbornej praxe

Znalosti uvedené v tejto sekcii mi počas vykonávania odbornej praxe chýbali, a teda musel som ich doplniť samoštúdiom.

Aj keď som počas štúdia nadobudol skúsenosti v programovacom jazyku Java jednalo sa len o základ a v porovnaní s reálnou praxou ich bolo málo. Taktiež mi chýbali mi znalosti v jazyku Java EE, ktorý bol potrebný pre tento projekt. Našťastie doštudovať potrebné informácie nebol veľký problém vďaka dokumentácii, ktorá je dostupná online.

Určité nedostatky sa naskytli aj pri databáze MySQL, ale vďaka absolvovaní predmetov v škole, ktoré sa touto problematikou zaoberali som sa oboznámil s riešením problémov za relatívne krátku dobu. Určité nedostatky sa objavili až keď som začal zachádzať do hlbšej problematiky v oblasti HTML a JavaScriptu, ktoré sa nepreberali v predmete Vývoj internetových aplikácii.

Môj Najväčší nedostatok bol vo frameworku Spring Boot a šablónovacom systéme Thymeleaf. Doštudovanie týchto dvoch nedostatkov zabralo najviac času, avšak ako som sa s nimi viac a viac zoznamoval stávali sa zrozumiteľné.

Taktiež softvére Docker som vôbec nepoznal, ale naučiť sa s ním pracovať nebol problém, keďže je dostupná výborná dokumentácia online.

Kapitola 7

Závěr

Absolvovanie tejto odbornej praxe hodnotím veľmi kladne a som vďačný, že univerzita umožňuje takýmto spôsobom študentom získať praktické skúsenosti v oblasti ich štúdia.

Možnosť vykonávať prax v tejto spoločnosti bolo veľmi prínosné, keďže sa jedná o spoločnosť pôsobiacu vo viacerých krajinách, v ktorej som získal mnoho skúseností, ktoré by som inak získaval až po dokončení štúdia na VŠB.

Som vďačný, že mi spoločnosť Marelli Automotive Lighting Jihlava (Czech Republic) s.r.o. darovala týmto spôsobom mnoho skúseností, ktoré nebudú zbytočne získané. Získal som skúsenosti s novými technológiami a reálnym priebehom procesom vývoja softvéru vo firme.

Výstupom práce, ktorú som vykonával je softvér, ktorý sa bude používať na správu licencií v oddelení R&D.

Daný informačný systém dovoľuje do budúcna rozšírenie o mobilnú alebo desktopovú platformu tohto systému.

Literatura

1. *MySQL* [online] [cit. 2021-02-10]. Dostupné z : https://www.hostinger.com/tutorials/what-is-mysql.
2. *Docker* [online] [cit. 2021-02-10]. Dostupné z : https://www.freecodecamp.org/news/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b/.
3. *Java EE* [online] [cit. 2021-02-11]. Dostupné z : https://www.javatpoint.com/java-ee.
4. *Spring Boot* [online] [cit. 2021-02-13]. Dostupné z : https://stackify.com/what-is-spring-boot/.
5. *Thymeleaf* [online] [cit. 2021-02-13]. Dostupné z : https://medium.com/thymeleaf-basics-concepts/thymeleaf-1ef952db0740.
6. *JavaScript* [online] [cit. 2021-02-14]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.
7. *HTML* [online] [cit. 2021-02-14]. Dostupné z : https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics.
8. *Kaskadové-štýly* [online] [cit. 2021-02-14]. Dostupné z : https://www.tutorialspoint.com/css/what_is_css.htm.
9. *SCSS* [online] [cit. 2021-02-14]. Dostupné z : https://www.dailysmarty.com/posts/what-is-scss.
10. *jQuery library* [online] [cit. 2021-02-15]. Dostupné z : https://www.javatpoint.com/what-is-jquery.
11. *Bootstrap* [online] [cit. 2021-02-15]. Dostupné z : https://careerfoundry.com/en/blog/web-development/what-is-bootstrap-a-beginners-guide/.
12. *InteliJ* [online] [cit. 2021-02-16]. Dostupné z : https://www.techopedia.com/definition/7755/intellij-idea.
13. *Model–view–controller* [online] [cit. 2021-02-18]. Dostupné z : https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm.
14. *Maven* [online] [cit. 2021-02-19]. Dostupné z : https://maven.apache.org/pom.html.

15. *Spring Boot starter web* [online] [cit. 2021-02-20]. Dostupné z : https://www.javatpoint.com/spring-boot-starter-web.

16. *Spring Boot starter data JPA* [online] [cit. 2021-02-20]. Dostupné z : https://www.javatpoint.com/spring-boot-starter-data-jpa.

17. *MySQL connector Java* [online] [cit. 2021-02-20]. Dostupné z : https://stackoverflow.com/questions/46995281/maven-dependencies-springboot-and-mysql.

18. *Spring Boot starter Thymeleaf* [online] [cit. 2021-02-20]. Dostupné z : https://www.javatpoint.com/spring-boot-thymeleaf-view.

19. *Spring Boot starter security* [online] [cit. 2021-02-20]. Dostupné z : https://www.javatpoint.com/spring-boot-starters.

20. *Spring Boot starter mail* [online] [cit. 2021-02-20]. Dostupné z : https://www.javatpoint.com/spring-boot-starters.

21. *Spring Boot devtools* [online] [cit. 2021-02-20]. Dostupné z : https://docs.spring.io/spring-boot/docs/1.5.16.RELEASE/reference/html/using-boot-devtools.html.

22. *Super CSV* [online] [cit. 2021-02-20]. Dostupné z : http://super-csv.github.io/super-csv/index.html.

23. *Identifikátory* [online] [cit. 2021-02-24]. Dostupné z : https://www.baeldung.com/hibernate-identifiers.

24. *Relačné mapovanie* [online] [cit. 2021-02-26]. Dostupné z : https://hellokoding.com/jpa-one-to-many-relationship-mapping-example-with-spring-boot-maven-and-mysql/.

25. *Zoraďovanie* [online] [cit. 2021-02-28]. Dostupné z : https://www.logicbig.com/tutorials/java-ee-tutorial/jpa/order-by-annotation.html.

26. *Relačné mapovanie* [online] [cit. 2021-03-01]. Dostupné z : https://hellokoding.com/jpa-many-to-many-relationship-mapping-example-with-spring-boot-maven-and-mysql/.

27. *JoinTable anotácia* [online] [cit. 2021-03-02]. Dostupné z : https://docs.oracle.com/javaee/7/api/javax/persistence/JoinTable.html.

28. *Transient anotácia* [online] [cit. 2021-03-02]. Dostupné z : https://javabydeveloper.com/using-transient-in-spring-boot-examples/.

29. *Rozhranie UserDetails* [online] [cit. 2021-03-04]. Dostupné z : https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/UserDetails.html.

30. *Autowired anotácia* [online] [cit. 2021-03-05]. Dostupné z : https://www.journaldev.com/2623/spring-autowired-annotation.

31. *Configuration anotácia* [online] [cit. 2021-03-06]. Dostupné z : https://www.journaldev.com/21033/spring-configuration-annotation.

32. *EnableWebSecurity anotácia* [online] [cit. 2021-03-07]. Dostupné z : https://www.baeldung.com/spring-boot-security-autoconfiguration.

33. *Controller anotácia* [online] [cit. 2021-03-08]. Dostupné z : https://zetcode.com/springboot/controller/.

34. *Requestmapping anotácia* [online] [cit. 2021-03-09]. Dostupné z : https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html.

35. *InitBinder anotácia* [online] [cit. 2021-03-10]. Dostupné z : https://www.logicbig.com/tutorials/spring-framework/spring-web-mvc/spring-custom-property-editor.html.

36. *RequestParam anotácia* [online] [cit. 2021-03-11]. Dostupné z : https://www.javatpoint.com/spring-mvc-requestparam-annotation.

37. *PathVariable anotácia* [online] [cit. 2021-03-12]. Dostupné z : https://zetcode.com/spring/pathvariable/.

38. *GetMapping anotácia* [online] [cit. 2021-03-13]. Dostupné z : https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/GetMapping.html.

39. *PostMapping anotácia* [online] [cit. 2021-03-14]. Dostupné z : https://www.baeldung.com/spring-new-requestmapping-shortcuts.

40. *Scheduled anotácia* [online] [cit. 2021-03-15]. Dostupné z : https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/scheduling/annotation/Scheduled.html.